

# Arquitectura de computadores

---

Pau Arlandis Martínez

## Sobre las normas de la asignatura

### Teoría

4 Prácticas (voluntarias). Solo puedes presentarte una vez.

- Entrada/Salida → Simulador 15%
- Memoria caché
- Pipeline de instrucciones } MC 88110 35%
- Multiprocesadores → C 25%

Las prácticas pueden sumar, juntas, hasta un punto más en la nota de teoría.

3 parciales

- Tema 1 → 30%
- Tema 2 → 40%
- Tema 3 y 4 → 40% (Posibilidad de recuperar tema 1 [25%] o tema 2 [35%])

Debe obtenerse una nota mínima de 2 puntos en cada uno para que hagan media.

### Práctica

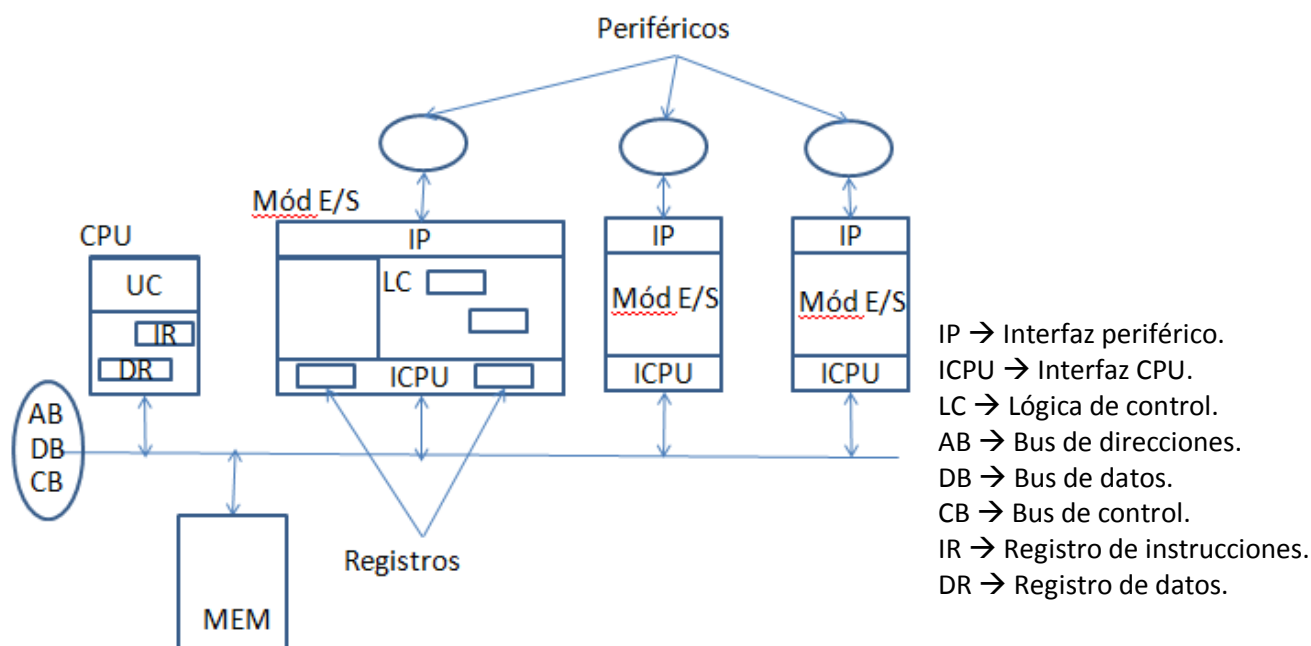
1 proyecto (hasta la semana 9)

- Entrada/ Salida.
- Examen (al terminar, semana 9).
- Memoria escrita.

### Nota final

Siempre que ambas partes (teoría y proyecto) estén aprobadas (con más de un 5 cada una) la nota final se calcula mediante la fórmula:

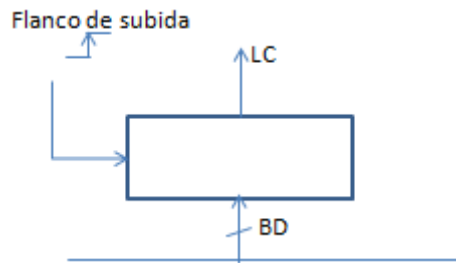
$$0.7 \times nota_{teoría} + 0.3 \times nota_{proyecto}$$



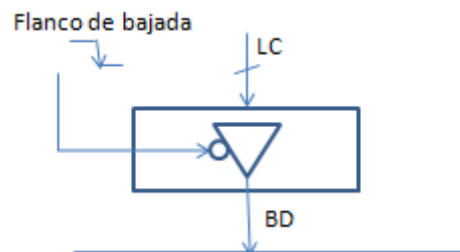
Lo único que controlamos son los registros de la Interfaz con la CPU. Estos son de tres tipos:

### Registros de datos

- **Registros de salida.** Un conjunto de biestables que nos permiten sacar información desde el bus de datos a la lógica de control.



- **Registro de entrada.** Saca información de la LC y la vuelca en el bus de datos. La salida se controla mediante un buffer triestado con estados 0, 1 y nada, vacío.



### Registros de estado

Son los que indican el estado del periférico y permiten el control de errores.

### Registros de control

Se encargan del control de todos los componentes y registros del módulo.

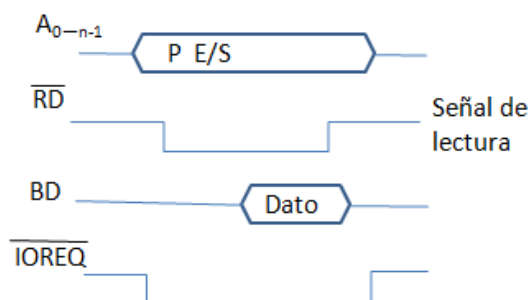
### Instrucciones de los módulos

Existen dos formas de dirigir el control de los módulos y de mapear sus direcciones.

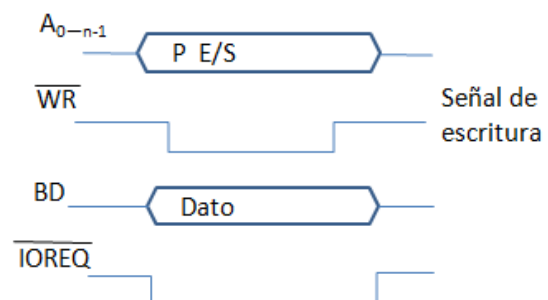
#### Mapas separados

Para controlar los módulos E/S con este método es necesario establecer un par de ciclos nuevas que permitan la entrada y la salida desde el procesador a los módulos.

#### Ciclo de entrada

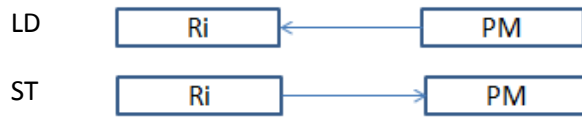


#### Ciclo de salida

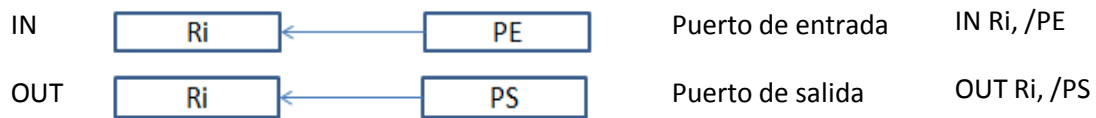


Para que la memoria codifique, lea o modifique la información del bus se activa la señal  $\overline{\text{MEMRQ}}$ , para los módulos de E/S haremos igual. Crearemos la señal  $\overline{\text{IOREQ}}$ , si no está activa no hace nada la CPU, pero si lo está inicia la ejecución en el módulo.

A nivel de instrucciones sucede lo mismo, existen dos instrucciones para acceder a la memoria existen dos instrucciones:

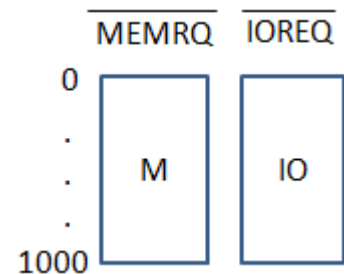


De forma paralela trabajaremos con los módulos, en los que existirán dos instrucciones:



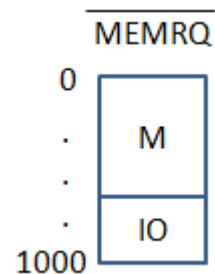
En este caso tendremos dos rangos de direcciones diferentes:

- 0-1000 (por ejemplo) con  $\overline{\text{MEMRQ}}$  activada.
- 0-1000 con  $\overline{\text{IOREQ}}$  activada.

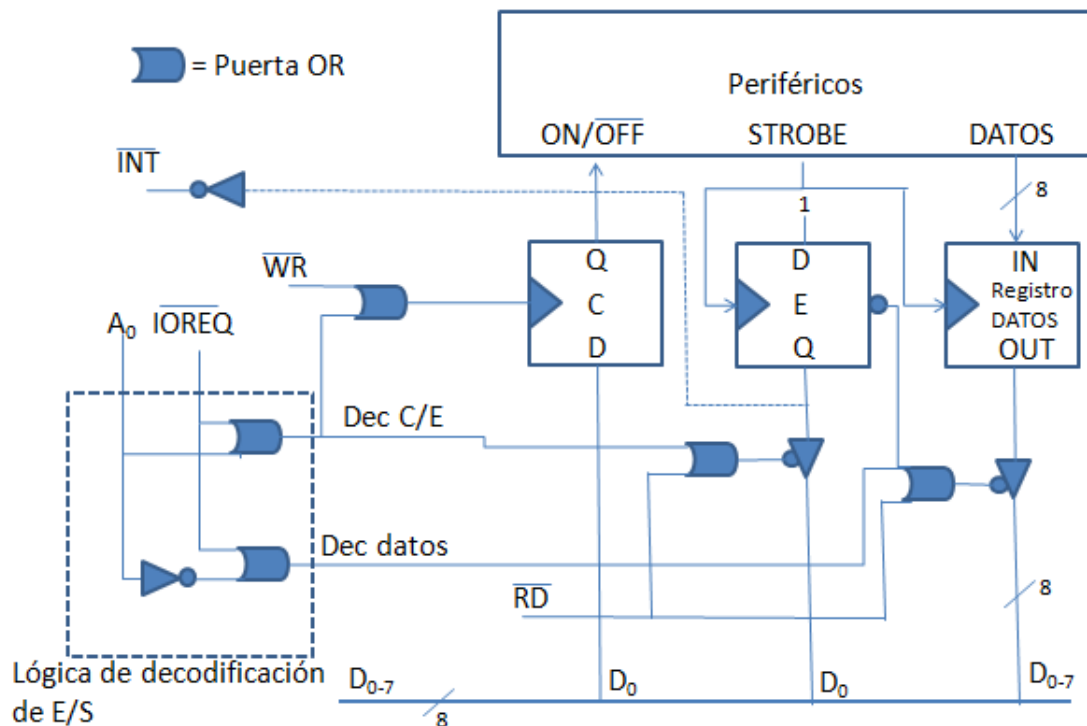


### Mapeado en memoria

En este caso solo existe un rango de direcciones ya que no se dispone de señal  $\overline{\text{IOREQ}}$ , entonces no existen las instrucciones IN y OUT, solo las de direccionamiento de memoria. La solución que se toma es reservar un subrango de direcciones de la memoria para las de E/S. Debe tomarse un rango al principio o al final para mejorar la eficiencia.



## Ejemplo de módulo de E/S



Vamos a suponer que todas las direcciones son para este periférico ya que es único. Las direcciones que tengan un 0 en el bit menos significativo ( $A_0$ ) hacen referencia al registro de control (C) o al de estado (E). Si tienen un 1, hacen referencia al de datos (D).

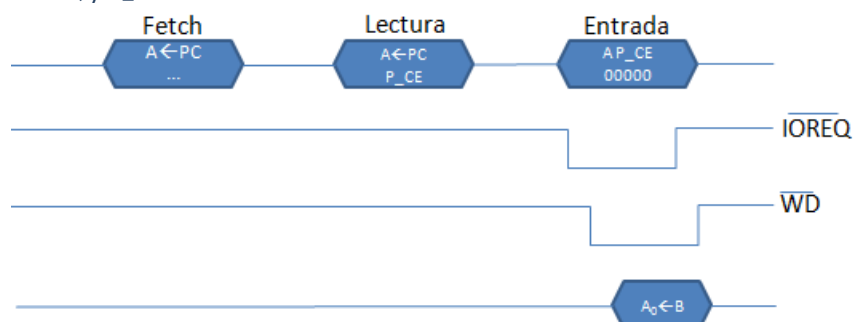
- $xx...x_0 \rightarrow P\_CE$ , por ejemplo  $\rightarrow 000...0$
- $xx...x_1 \rightarrow P\_D$ , por ejemplo  $\rightarrow 000...1$

Para cada registro existen, pues, condiciones para su lectura o escritura:

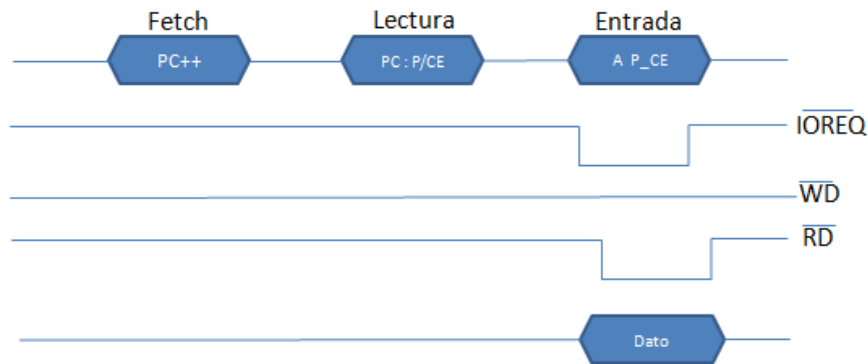
- C. Se activa si tenemos una dirección par ( $A_0=0$ ), si la señal de entrada/salida está activa (IOREQ) y si la señal de escritura (WR) está activa.
- E. Se activa si tenemos una dirección par, si la señal de entrada/salida está activa y si la señal de lectura (RD) está activa.
- D. Se activa si tenemos una dirección impar ( $A_0=1$ ), si la señal de entrada/salida está activa y su la señal de lectura está activa.

### Cronograma

in .R0, /P\_CE



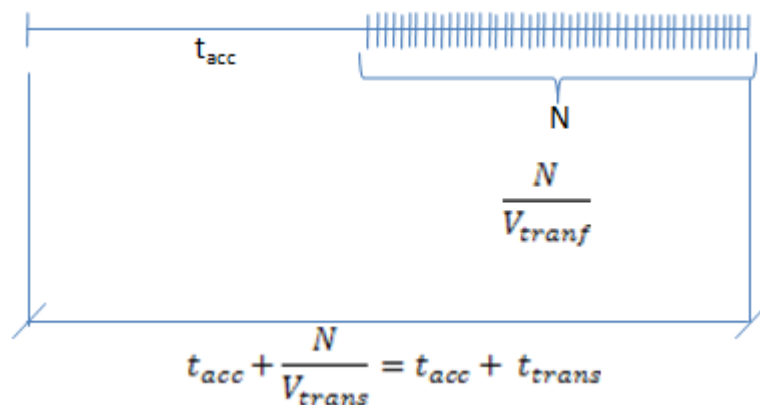
out .R1, /P\_CE



En este caso el dato es el registro de estado.

## Operación de E/S

Operación de intercambio de un bloque de datos de  $n$  bytes entre un periférico y la memoria. El tiempo de acceso es el tiempo que tarda en llegar el primer dato del bloque a la memoria, a partir de entonces comienzan a transmitirse a una determinada velocidad de transferencia.



## Técnicas de E/S

Las técnicas de E/S nos permiten saber cómo organizar el intercambio de datos y las instrucciones de entrada/salida para aprovechar mejor el trabajo de la CPU y así ejecutar más programas.

Existen tres técnicas:

- E/S programada.
- E/S por interrupciones.
- E/S por DMA.

Ordenadas por orden de uso de la CPU, de mayor a menor.

Tienen en común 4 fases básicas:

- Iniciar (I). Prepara la operación. Prepara el buffer de E/S.

- Sincronizar (S). Comprueba si el siguiente dato está listo. Si lo está comienza la transmisión.
- Transferir (T). Reg. Datos → M ó M → Reg. Datos. Vuelve a la fase S y espera a que otro dato esté listo.
- Finalizar (F). Dialoga con el periférico para conocer su estado final y qué debe hacer a continuación.

Visto sobre una línea temporal:



## Programada

### Ejemplo

#### Datos del procesador

- 100 MIPS (Millones de Instrucciones Por Segundo)
- $t_{acc} = 1ns$
- $V_{transf} = 10^6 B/seg$
- Dir\_CE → Dirección de Control y Estado
- Dir\_D → Dirección de datos
- ON = 1111 ; 0001
- OFF = 0000
- LISTO = 00001 (X X X X V)

Escribir en ensamblador cada una de las fases básicas de entrada/salida:

```

I { LD .R1, #Dir_Alm_M      ;Dirección de almacenamiento de memoria.
  LD .R2, #1000            ;Contador de datos
  LD .R0, #01
  OUT .R0, /Dir_CE         ;Almacena el contenido de R0(1) en el
                           ;módulo y se conecta
S { SIG: IN .R0, /Dir_CE    ;xxxxV
  AND .R0, #0001           ;0000V
  CMP .R0, #LISTO          ;Listo es V
  BNE $SIG                 ;si no son iguales mantiene el bucle hasta
                           ;que lo sea.
T { IN .R0, /Dir_datos      ;Leemos el dato del periférico
  ST .R0, [.R1++]           ;Guardamos el dato en la memoria.
  DEC .R2                  ;Decrementamos el contador de datos porque
                           ;ya ha llegado uno.
  BNZ $SIG                 ;Si no se han enviado todos los datos
                           ;continúa el bucle.
F { LD .R0, #0000
  OUT .R0, /Dir_CE

```

$$t_{opES} = t_{INI} + t_{acc} + N/V_{transf} + t_{TE} + t_F \cong t_{acc} + N/V_{transf} = 1ms + \frac{1000}{10^6} \cong 2ms$$

Pero de este tiempo solo hacemos trabajo útil durante el tiempo de transferencia elemental ( $t_{TE}$ )

$$t_T = 1000 \times 4 \text{ Ins} = 1000 \times \frac{4}{10^8} = 40 \mu\text{s es el tiempo de transferencia elemental}$$

$\frac{4}{10^8}$  es la inversa de MIPS  $\frac{1}{100 \times 10^6}$  por 4 Instrucciones.

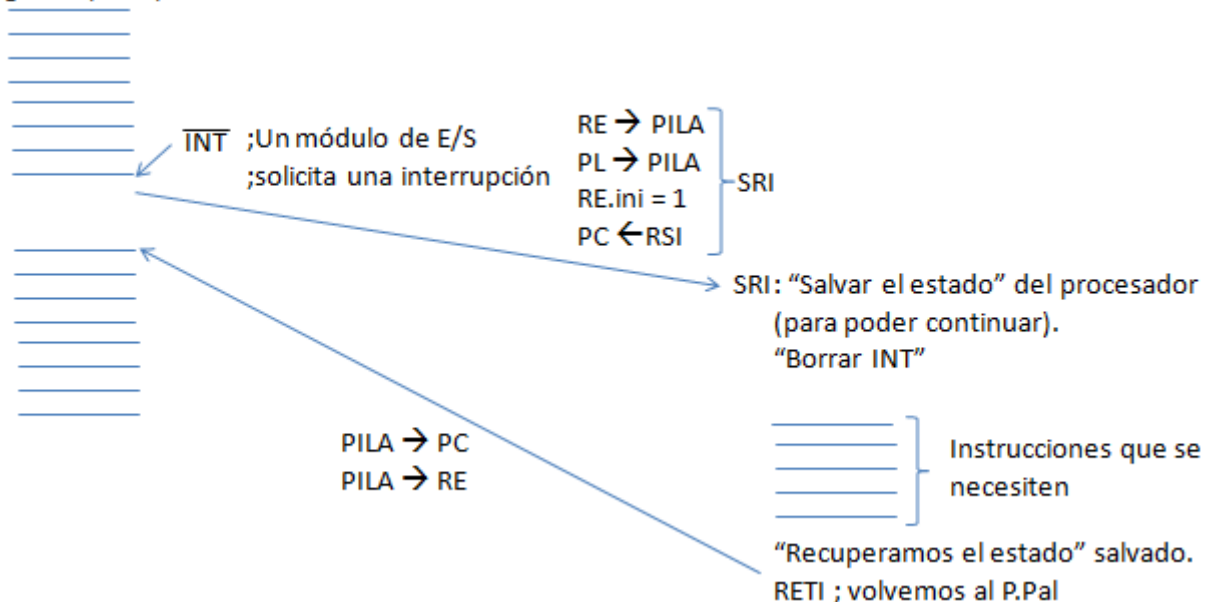
Esto quiere decir:

$$\frac{40}{2000} \times 100 = 2\% \text{ es el porcentaje de tiempo del trabajo útil.}$$

$100 - 2 = 98\%$  es el tiempo que pasamos en el bucle de sincronización.

## Interrupciones

Programa principal



Al volver de la interrupción es importante que todos los registros sean idénticos a como estaban antes de la interrupción o podremos cometer un error y que el programa no ejecute correctamente.

## Ejemplo

```
LD .R1, #Dir_alm_mem      ; Dirección de almacenamiento de
                           ; memoria.
ST .R1, /Dir_dir_alm_mem  ; Guardamos la dir_alm_mem que
                           ; desconocemos en una variable global
                           ; conocida por todos y que no pueda
                           ; variar otro proceso.

LD .R2, #1000
ST .R2, /Dir_contador     ; Ídem con el contador.
LD .R0, #01               ; "ON".
OUT .R0, /Dir_CE          ; Comienza a funcionar el periférico.
BR $Planificador          ; Cedemos el control al OS.
```



```

; Tras las instrucciones que sean que nada tienen que ver con el
; módulo.
RSI: PUSH .R0
    PUSH .R1
    PUSH .R2                                ; Almacenamos en la pila el
                                           ; estado actual del programa.
    LD .R1,/Dir_dir_alm_mem                ; Recuperamos la dirección donde
                                           ; podemos almacenar la memoria.
    LD .R2,/Dir_contador                    ; Recuperamos el contador.
    IN .R0,/Dir_datos
    ST .R0,[.R1++]
    ST .R1,/Dir_dir_alm_mem
    DEC .R2                                ; Decrementamos R2.
    CALLZ $FIN                              ; Si R2 es 0, hemos terminado la
                                           ; interrupción y salta a la
                                           ; rutina que la finaliza.

    ST .R2,/Dir_contador
    POP .R2
    POP .R1
    POP .R0
    RETI

FIN: LD .R0,#00                            ; "OFF"
    OUT .R0,/Dir_CE
    CALL "Comp_errores"                    ; Llamamos al módulo E/S para saber
                                           ; si ha habido errores.
    CALL "FINOP_SI"                        ; Informa al OS de que la rutina de
                                           ; interrupción ha terminado.

    RET

```

Suponiendo:

$$100 \text{ MIPS} \rightarrow t_{\text{inst}} = \frac{1}{10^8} = 10 \mu\text{s} \quad V_{\text{transf}} = 10^6 \text{ b/s} \quad t_{\text{acc}} = 1 \text{ ms}$$

$$t_{\text{opES}} = t_{\text{ini}} + t_{\text{acc}} + \frac{1000}{10^6} + t_{\text{SRI}} + t_{\text{RSI}} + t_{\text{FIN}} = 1 \text{ ms} + 1 \text{ ms} = 2 \text{ ms}$$

$$t_r = 1000 (t_{\text{SRI}} + t_{\text{RSI}}) = 1000 (1 \text{ ns} + 15 \text{ ns}) = 160 \mu\text{s}$$

Es decir, cuatro veces más que en la programada. Ahora sabemos por qué se utiliza la técnica "por interrupciones":

$$t_{\text{CPULIBRE}} = 2000 - 160 \mu\text{s} = 1840 \mu\text{s}$$

$$\%CPU_{\text{LIBRE}} = \frac{1840}{2000} \times 100 = 92\%$$

Este porcentaje ya no se gasta en hacer bucles, sino que está ocupado en otros procesos, esa es la gran diferencia.

$$\%CPU_T = \frac{160}{2000} \times 100 = 8\% \text{ de gasto de la CPU}$$

## Problemas

- **Conexionado.** ¿Cómo conectar todos los módulos (ya que siempre hay más de uno) y sus respectivas líneas de petición de interrupción?
- **Identificación.** ¿Cómo se sabe a quién pertenece una interrupción?
- **Localización.** ¿Qué subrutina de servicio debe utilizarse en cada interrupción?
- **Prioridades.** ¿Cómo sabemos a quién dar prioridad cuando existen varios periféricos interrumpiendo?
- **Anidamiento de rutinas.**

## Soluciones

### Conexionado

La solución más utilizada y eficiente es conectar cada línea de interrupción en una sola hacia el CPU.



En cada conector existe una puerta llamada de colector abierta (en CMOS) o de drenador abierto (en NMOS) que permite que este cable de interrupción funcione como un OR cableado. Siempre que uno (o más) periféricos estén

activos el cable está activo y solo pasa a estar inactivo cuando TODOS los periféricos lo estén.

### Identificación y localización

#### ~mediante muestreo

Solo hay una rutina de tratamiento de interrupción (RTI) que pregunta uno a uno a todos los periféricos en el orden de prioridad dado. Cuando encuentre una petición de interrupción salta a la subrutina del módulo que la pedía, por tanto resuelve ya el problema de la prioridad además del de la localización e identificación. Sin embargo no permite anidamiento de rutinas y no es muy eficiente.

#### Vectorización

Esta forma de identificación conceptualmente dota a la CPU de una forma de preguntar quién está enviando la interrupción. Para ello hace uso de una nueva señal, en el ciclo de bus, de reconocimiento de interrupción (SRI), la señal INTA. Esta señal pide al dispositivo que se identifique. Para poder implementarlo es necesario que el módulo posea un vector de identificación.

Este vector debe ser sencillo de implementar.

### Prioridades

A parte de las ya comentadas en el apartado anterior:

#### Gestor centralizado

Cada módulo tiene una señal de interrupción y una de reconocimiento. Así es sencillo conocer la prioridad y la identificación de cada módulo. Esto también permite tener flexibilidad en la

asignación de prioridades. Sin embargo al estar el sistema cableado no es escalable, no pueden añadirse más módulos que los que ya existen.

También permite anidamiento, pero no es fácil de implementar.

### Gestor encadenado

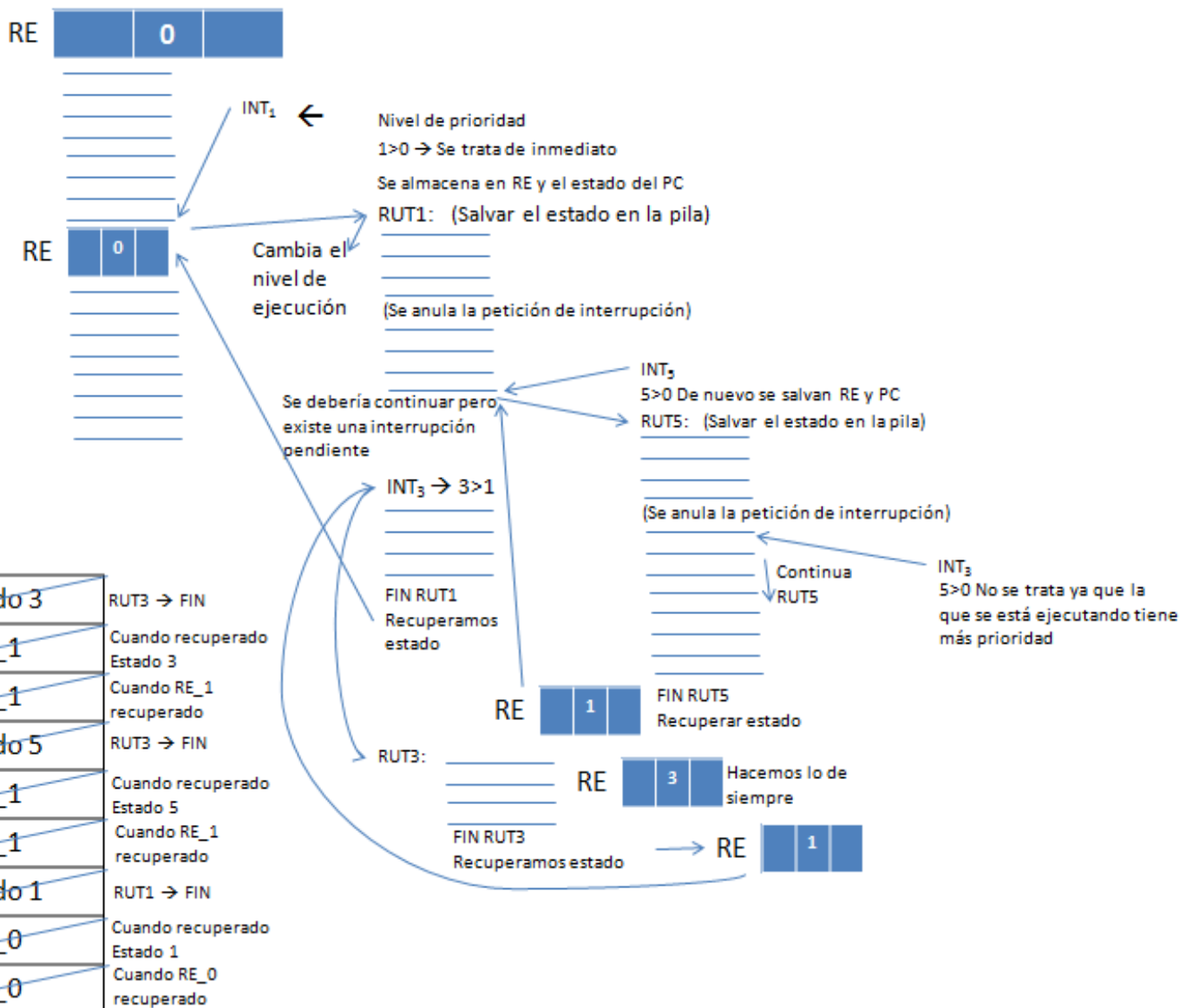
En este caso, la señal de reconocimiento es única y está conectada a todos los módulos de forma anidada. La señal pasa por todos los módulos en orden de prioridad y para en el primero que esté activo. La asignación de prioridades se establece por construcción y por tanto es fija, sin embargo, pueden añadirse más módulos.

### Híbrido

Combina ambas soluciones.

### Anidamiento

#### Ejemplo de ejecución



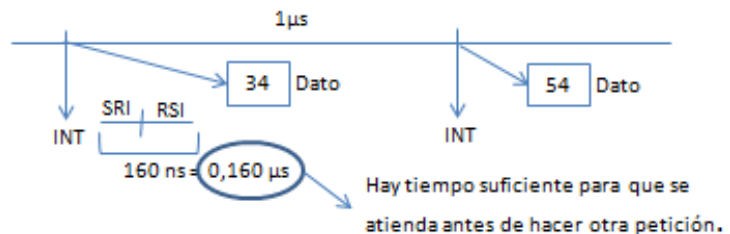
Cuando llega una petición de interrupción de nivel igual o inferior al proceso que se está ejecutando se evalúa únicamente cuando esta última ha terminado.

Si la petición es de nivel mayor se ejecuta de inmediato.

### Ejercicio

#### Computador

- $10^6$  b/seg
- 100 MIPS  $\rightarrow t_{ins} = 10\text{ns}$
- $t_{SRI} = 1$  Instrucción
- $t_{RSI} = 15$  Instrucciones



$$\text{Frecuencia de interrupción} = \frac{10^6 \text{ b/s}}{1 \text{ Ins/B}} = 10^6 \text{ Ins/s}$$

$$t_{\text{entre\_ins}} = \frac{1}{10^6 \text{ Ins/s}} = 1 \mu\text{s}$$

$t_{\text{respuesta}} =$    
 depende de CPU, no es algo conocido.   
 $t_{\text{res\_max}} \rightarrow$  Es lo único que podemos saber.   
 Es el tiempo máximo que puede esperar una interrupción para no perder datos.

$$t_{\text{res\_max}} = t_{\text{entre\_ins}} - (t_{SRI} + t_{RSI})$$

Pero puede ser que el CPU esté haciendo otra cosa o atendiendo otra petición y se posponga.

Puede incluso tratarse después de que llegue otra interrupción y se evalúe un dato distinto, perdiendo datos.

Existen, pues, límites al tratamiento por interrupciones que dependen, por un lado, de lo que tarda el periférico en emitir una interrupción. Por otro lado, de la velocidad a la que se procesen las instrucciones y el mínimo de instrucciones de la rutina RSI.

### Consumo

$\text{Consumo}_{\text{CPU}} = \text{Frec}_{\text{int}} \cdot (t_{SRI} + t_{RSI}) = 10^6 \cdot 16 = 16 \text{ MIPS}$  es el consumo de IPS que gasta el periférico.

Si el consumo fuese mayor que las IPS que es capaz de ejecutar la CPU (en este caso,  $\text{consumo}_{\text{CPU}} > 100 \text{ MIPS}$ , no se da), el procesador no sería capaz de hacer funcionar el periférico.

Sumando el consumo de todos los periféricos tendremos el consumo total. Para que un sistema por interrupciones pueda funcionar, el consumo total debe ser menor que la velocidad del procesador.

$$\text{Cons}_{P1} = C_1 = F_{I1} \cdot (t_{SRI} + t_{RSI})$$

$$\text{Cons}_{P2} = C_2 = F_{I2} \cdot (t_{SRI} + t_{RSI})$$

•  
•  
•

$$+ \text{Cons}_{Pn} = C_n$$

---


$$\Sigma \leq MIPS$$

### Criterio de priorización

Existen múltiples criterios para asignar prioridades a las interrupciones, entre ellas las más importantes son:

- **Frecuencia de interrupciones.** Cuantas más interrupciones haga en un tiempo determinado más prioridad.
- **Urgencia de las instrucciones.** Periféricos que solo interrumpan en momentos críticos tienen alta prioridad. Se denominan interrupciones no enmascarables, para ellos existen peticiones de interrupción especiales.

### Resolviendo problemas con las interrupciones

$$\begin{array}{l} t_{\text{respuesta}} \rightarrow \\ t_{\text{procesamiento}} \rightarrow \end{array} \left| \begin{array}{l} \text{Fre}_{C_{INT}} = \\ \end{array} \right. \begin{array}{l} V_{\text{transf}} \rightarrow \text{Depende del periférico. No podemos modificarla.} \\ t_{INT} \text{ (Número de instrucciones necesarias para satisfacer las} \\ \text{interrupciones)} \rightarrow \text{Podemos modificarlo hasta cierto punto.} \end{array}$$

Entonces, ¿Cómo podemos modificar un módulo de E/S y reducir su frecuencia?

Una de las cosas que podemos hacer es aumentar el número de bytes que se transfieren en cada operación. En vez de tener un registro donde se almacenan los datos tendremos varios registros.



Tenemos varias posibilidades:

- Número de registros que tengamos.
- Capacidad de cada registro.

Por ejemplo:

- 4x1 → 4 registros de 1 byte → 4 B
- 1x4 → 1 registro de 4 bytes → 4 B
- 4x4 → 4 registros de 4 bytes → 16 B

Cuanta más capacidad de memoria temporal tenga el módulo, más eficiente será el sistema.

Teniendo en cuenta este nuevo factor:

$$t_{opES} = t_{INI} + t_{acc} + \frac{N}{V_{transf}} + t_{int} + t_{FIN}$$

En este caso  $t_{int}$  será un poco más grande al ser la RSI más complicada. Sin embargo,  $t_{opES}$  apenas cambia.

$$t_{CPU} = t_{int} + \frac{N}{NR \times LR} \times (t_{SRI} + t_{RSI}) + t_{FIN}$$

Donde NR = Número de registros y LR = Longitud de registro.

Esta parte cambia considerablemente ya que disminuye el número de interrupciones, pues antes solicitaba una interrupción por cada byte y ahora cuando se llena la memoria temporal. Existen módulos que envían dicha interrupción cuando reciben el primer byte. En este caso, el número de interrupciones es  $\frac{N}{NR \times LR}$ .

### Tiempo de salida

$$t_{opW} = t_{INI} + t_{acc} + \frac{N}{V_{transf}} + t_{FIN}$$

En el caso de una operación W se termina cuando llega el último dato.

## Acceso Directo a Memoria (DMA)

Básicamente la técnica DMA deja en manos del módulo de E/S, tanto la sincronización (como en la técnica por interrupciones) como la transferencia. Solo se avisa a la CPU al finalizar.

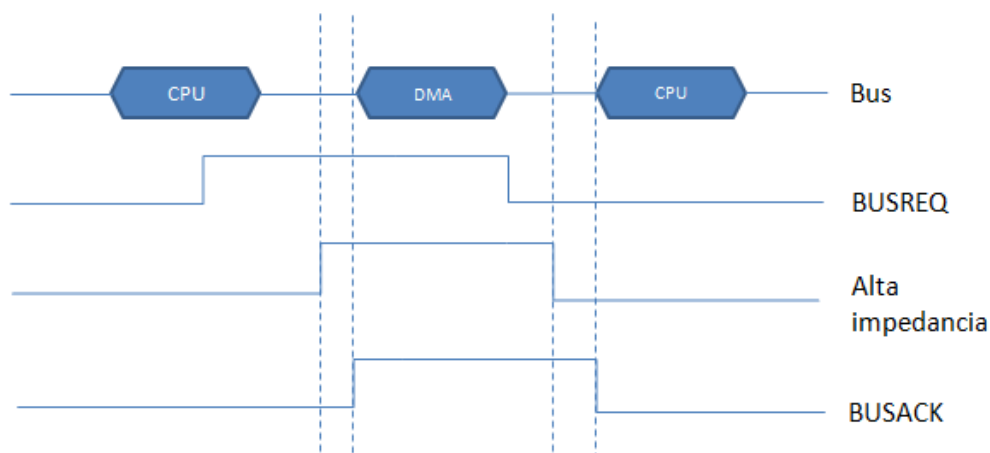
Para ello debemos realizar modificaciones en el módulo de E/S, hacerlo más complejo:

- Añadir dos registros más, el registro de direcciones y el registro de contador.
- Generar las señales de dirección y control (RD, WR, MEM) entre otras.
- Necesitamos un incrementador, un decrementador y un comprobador.
- Introduciremos dos señales nuevas, para solicitar el bus cuando el módulo necesite recibir o enviar información al mismo.

### Petición de bus

Robo de ciclo aislado

Byte a byte. 1 registro de 1 byte.



$$t_{opES} = t_{INI} + t_{acc} + \frac{N}{V_{transf}} + t_{DMA} + t_{FIN}$$

$$t_{CPU} = t_{INI} + N \underbrace{(t_{PCD} + t_{CM})}_{t_{DMA}} + \underbrace{(t_{SRI} + t_{RSI})}_{FIN}$$

Donde N es el número de operaciones de DMA:  $N_{DMA} = \frac{N}{NR \times LR}$  y  $t_{PCD}$  es el tiempo del protocolo de concesión de datos.

Si el registro tiene una longitud mayor a un byte o tiene varios bytes, el número de operaciones de DMA será menor.

### Robo de ciclo en ráfagas

Utiliza tantos ciclos de DMA como sean necesarios, reduciendo el número de interrupciones en el bus.



$$t_{CPU} = t_{INI} + \frac{N}{NR \times LR} \underbrace{(t_{PCD} + NR \times t_{CM})}_{t_{DMA}} + \underbrace{(t_{SRI} + t_{RSI})}_{FIN}$$

$$N^{\circ} DMA = \frac{N}{NR \times LR}$$

Si la operación es de salida, la primera operación de DMA se hace de forma paralela al  $t_{acc}$ .

## Problemas

### Ejercicio primero

Un procesador de 32 bits; 40 MIPS ;  $t_{CM} = 20$  ns

a)

SRI: FETCH: Si  $i > \max \wedge I > RE.BMI$  (Donde, i = interrupción; max = máxima de todas las INT; RE.BMI = Biestable de máscara de INT).

```

*PC → PILA
*RE → PILA
RE.BMI ← i
RE.S ← 1 ; Pasamos a modo supervisor
CRI: *INTA i ; BD → vector
*PC ← M(RBTv + vector x 4) ; RBTv (Registro Base de Tabla
; de Vectores.
(FETCH)
SINO: (FECH)
  
```

Todas las operaciones con asterisco son las que gastan tiempo, por tanto:

$$t_{SRI} = 4 \times 20 \text{ ns} = 80 \text{ ns}$$

b) Tenemos los datos:

- bloques = 64  $\rightarrow$  1536 b     $N = 1536 \text{ b} \rightarrow 1536 \cdot 8 \text{ bits}$
- RD = 32 bits     $FR_{INT} \rightarrow \frac{32 \text{ bits}}{10^7 \frac{\text{b}}{\text{s}}} = 312500 \text{ Hz (también int por segundo)}$
- $V_{TRANSF} = 10^7 \text{ bits/s}$      $t_{TRANSF} = \frac{N}{V_{TRANSF}}$
- $t_{INI} = 50 \text{ l}$
- $t_{RSI} = 25 \text{ l}$

Se pide el número de instrucciones.

$$C_{CPU} = 312500 \times \left( \underbrace{80 \text{ ns}}_{t_{SRI}} + 25 \text{ Ins} \right) = 312500 (3,2 \text{ Ins} + 25 \text{ Ins}) = 8812500 \frac{\text{l}}{\text{s}}$$

$$= 8,8125 \text{ MIPS}$$

$$80 \times 10^{-9} \times 40 \times 10^6 = 3,2 \text{ Ins}$$

Entonces:

$$C_{CPULibre} = 40 \text{ MIPS} - 8,8125 \text{ MIPS} = 31,1875 \text{ MIPS}$$

Si se anula el  $t_{RSI}$  a 70 Ins y un procesador de 64 bytes con  $10^8 \text{ b/s}$  y velocidad de transferencia de bloques de 1024 b entonces:

$$FR_{INT} = \frac{10^8 \text{ bits/s}}{64 \times 8} = 195312,5 \text{ Hz}$$

$$C_{CPU} = 195312,5 \times (3,2 \text{ Ins} + 70 \text{ Ins}) = 14296875 \text{ Ins/s}$$

$$C_{libre} = 40 \times 10^6 - 14296875 = 25703125 \text{ Ins/s}$$

$$NI = \frac{1024 \times 8 \text{ bits}}{\underbrace{10^8 \text{ b/s}}_{t_{TRANSF}}} \times 25703125 = 2105 \text{ Ins}$$

### Robo de ciclo aislado

Con 1536 B

$$\%t_{CPU} = \frac{t_{CPU}}{t_{opES}} \times 100$$



$$\begin{aligned}
 t_{opES} &= t_{INI} + t_{acc} + \frac{1536 \times 8 \text{ b}}{\frac{10^9 \text{ b}}{\text{s}}} + t_{DMA} + (t_{SRI} + t_{RSI}) \\
 &= 50 \text{ Ins} + 0 + \frac{1236 \times 8}{10^9} \times 40 \times 10^6 + (10 \text{ ns} + 20 \text{ ns}) \times 10^{-9} \times 40 \times 10^6 \\
 &\quad + (3,2 + 70) \times \frac{1}{40 \times 10^6} = 15398 \mu\text{s}
 \end{aligned}$$

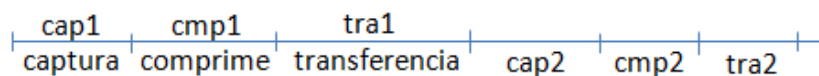
## Ejercicio segundo

### Computador

- 32 bits
- 400 MIPS
- Dedicado a la toma, compresión y transmisión de imágenes.
- Compresión de imágenes.
  - 512 KB → Se comprimen → 12 KB
  - El proceso de compresión dura  $15 \cdot 10^6$  Ins
- Transmisión por internet
  - $V_{TRANSF} = 100 \cdot 10^6 \text{ bits/s}$
  - RD → 32 bits
  - Tamaño de bloque = 1,5 KB
  - Rutina INI → 100 Ins
- Captura de imágenes
  - $t_{acc} = 0$
  - $V_{TRANSF} = 40 \cdot 10^6 \text{ bits/s}$
  - RD → 32 bits
  - INI = 50 Ins

Nuestro objetivo es tratar de capturar 25 imágenes por segundo, ¿con qué técnica de E/S podríamos hacerlo?

### E/S programada



Calculamos el tiempo (en instrucciones) que se tarda en capturar una imagen:

$$\begin{aligned}
 t_{cap} &= t_{INI} + t_{acc} + \frac{t_{amImg}}{V_{TRANSF}} = 50 \text{ Ins} + 0 + \frac{512 \text{ KB}}{40 \times 10^6} = 50 + \underbrace{\frac{512}{40 \times 10^6} \times 400 \times 10^6}_{\text{Ins}} \\
 &= 5242930 \text{ Ins}
 \end{aligned}$$

Calculamos el tiempo (en instrucciones) que se tarda en transmitir una imagen:

$$t_{tra} = \frac{12 \text{ KB}}{1,5 \text{ KB}} \times \underbrace{\left( t_{INI} + t_{acc} + \frac{1,5 \times 8}{100 \times 10^6} \times 400 \times 10^6 \right)}_{49152 \text{ Ins}} = 394016 \text{ Ins}$$

Si sabemos que el tiempo de compresión (en instrucciones) es:

$$t_{cmp} = 15 \times 10^6 \text{ Ins}$$

El tiempo total será la suma de estos tres valores:

$$t_{ima} = t_{cap} + t_{cmp} + t_{tra} = 20636946 \text{ Ins}/\text{Img}$$

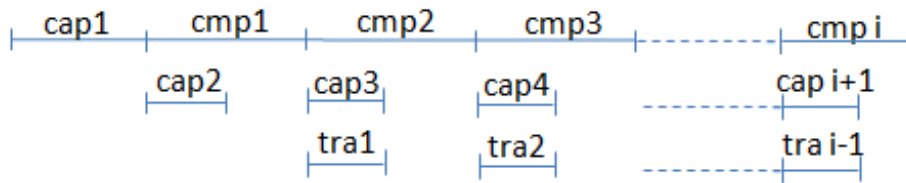
Entonces el procesador es capaz de procesar:

$$\text{ImgSeg} = \frac{400 \times 10^6}{20636946} = 19,38 \text{ Img/s}$$

Por tanto con una técnica de E/S programada no seríamos capaces de cumplir nuestro objetivo de procesar 25 imágenes por segundo.

### E/S por interrupciones

En este caso mientras se comprime una imagen, se captura la siguiente y se envía la anterior. Suponemos una Subrutina de Tratamiento de Interrupción de 5 Instrucciones (SRI = 5 Ins), y una Rutina de Servicio de Interrupción de 20 Instrucciones en el caso de la capturadora y de 35 instrucciones en el caso de la transmisión (RSI<sub>cap</sub> = 20 Ins y RSI<sub>tra</sub> = 35 Ins).



Primero debemos determinar la prioridad de cada operación, que se calcula mediante su RSI y la frecuencia de interrupciones.

$$\text{Frec}_{intTra} = \frac{100 \times 10^6 \text{ bits/seg}}{32 \text{ bits}} = 3,125 \times 10^6 \text{ Hz} \left( \text{Interrupciones} / \text{segundo} \right)$$

$$\text{Frec}_{intCap} = \frac{40 \times 10^6 \text{ bits/seg}}{4 \text{ bytes}} = 10 \times 10^6 \text{ Hz}$$

También calculamos el número de instrucciones por segundo de cada proceso:

$$\text{Consumo}_{tra} = 3,125 \times 10^6 \times (5 \text{ Ins} + 35 \text{ Ins}) = 125 \times 10^6 \text{ Ins/s} = 125 \text{ MIPS}$$

$$\text{Consumo}_{cap} = 10 \times 10^6 \times (5 \text{ Ins} + 20 \text{ Ins}) = 250 \times 10^6 \text{ Ins/s} = 250 \text{ MIPS}$$

En total 375 MIPS serían necesarios para capturar y transmitir al mismo tiempo, menos que los 400 MIPS que es capaz de procesar.

Una vez conocemos esto, es necesario calcular el número de instrucciones de cada proceso para saber cuántas imágenes procesa por segundo:

$$t_{cap} = 50 \text{ Ins} + \frac{512}{4} \times (5 \text{ Ins} + 20 \text{ Ins}) = 3276850 \text{ Ins}$$

$$t_{tra} = \frac{12}{1,5} \times \left( 100 \text{ Ins} + \frac{1,5}{4} \times (5 \text{ Ins} + 35 \text{ Ins}) \right) = 123680 \text{ Ins}$$

$$t_{cmp} = 15 \times 10^6 \text{ Ins}$$

En total:

$$t_{tot} = 18400030 \text{ Ins}$$

Por lo tanto:

$$\frac{400 \times 10^6}{18,400030 \times 10^6} = 21,73 \text{ Img/s}$$

Con una técnica de E/S por interrupciones seguimos sin cumplir nuestro objetivo de procesar 25 imágenes por segundo.

### **E/S por DMA**

Suponiendo que  $t_{DMA} = 10 \text{ ns}$  que son  $10 \times 10^{-9} \times 400 \times 10^6 = 4 \text{ Ins}$ . Entonces:

$$\begin{aligned} t_{cap} &= 50 \text{ Ins} + \frac{512}{4} \times 10 \text{ ns} + (5 \text{ Ins} + 20 \text{ Ins}) \\ &= 50 \text{ Ins} + \frac{512}{4} \times 4 \text{ Ins} + (5 \text{ Ins} + 20 \text{ Ins}) = 524363 \text{ Ins} \end{aligned}$$

$$t_{tra} = \frac{12}{1,5} \times \left( 100 \text{ Ins} + \frac{1,5}{4} \times 4 + (5 \text{ Ins} + 35 \text{ Ins}) \right) = 13408 \text{ Ins}$$

$$t_{cmp} = 15 \times 10^6$$

En total:

$$t_{tot} = 1537771 \text{ Ins}$$

Si necesitamos procesar 25 imágenes por segundo, ¿podemos utilizar DMA?

$$25 \times 1537771 = 388444275 \text{ Ins/s}$$

Que es menor de 400, por tanto: sí, sería posible. Sin embargo, observamos que el consumo del procesador con estas operaciones es:

$$\frac{388444275}{400 \times 10^6} \times 100 = 97,11 \%$$

Que es una cantidad excesivamente alta, este procesador estaría al límite de su capacidad. Por tanto sería recomendable tener un procesador capaz de procesar más instrucciones.



Apuntes de Arquitectura de Computadores by [Pau Arlandis Martínez](#) is licensed under a [Creative Commons Reconocimiento 3.0 Unported License](#).